Exercício sobre pytorch

Utilize o dataset 'datasetCarros.csv'.
Usando Pytorch, construa uma rede neural para prever a feature 'PrecoVenda'.

Use uma rede neural feed forward com duas camadas escondidas, com 50 neurônios cada.
Use o critério de perda MSELoss, otimizador Adam e learning rate = 0.001. Considere 10000 épocas.

In [ ]:
```
import torch
import numpy as np
import pandas as pd
```

In [ ]:
```
#Load dataset
data = pd.read_csv('datasetCarros.csv')
```

In [ ]:
```
data.head()
```

Out[ ]:

	Nome	Ano	PrecoVenda	PrecoAtual	KmRodado	TipoCombustivel	Trasmissao	Ov
0	ritz	2014	3.35	5.59	27000	Petrol	Manual	
1	sx4	2013	4.75	9.54	43000	Diesel	Manual	
2	ciaz	2017	7.25	9.85	6900	Petrol	Manual	
3	wagon r	2011	2.85	4.15	5200	Petrol	Manual	
4	swift	2014	4.60	6.87	42450	Diesel	Manual	

In [ ]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Carregar o dataset
dataset = pd.read_csv('datasetCarros.csv')

# Remover colunas não numéricas
dataset = dataset.drop(columns=['Nome', 'TipoCombustivel', 'Trasmissao'])

# Separar features e target
X = dataset.drop(columns=['PrecoVenda'])
y = dataset['PrecoVenda']

# Dividir dados em conjuntos de treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

```python
# Converter dados para tensores PyTorch
X_train_tensor = torch.tensor(X_train.values, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train.values, dtype=torch.float32).view(-1, 1)

X_test_tensor = torch.tensor(X_test.values, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.float32).view(-1, 1)

# Definir a arquitetura da rede neural
class Feedforward(torch.nn.Module):

    def __init__(self, input_size, hidden_size):
        super(Feedforward, self).__init__()

        self.input_size = input_size
        self.hidden_size  = hidden_size

        self.fc1 = torch.nn.Linear(self.input_size, self.hidden_size)
        self.fc2 = torch.nn.Linear(self.hidden_size, self.hidden_size)
        self.fc3 = torch.nn.Linear(self.hidden_size, 1)

    def forward(self, x):

        output = self.fc1(x)
        output = F.relu(output)

        output = self.fc2(output)
        output = F.relu(output)

        output = self.fc3(output)

        return output

# Parâmetros da rede neural
input_size = X.shape[1]
hidden_size = 50

# Instanciar o modelo
model = Feedforward(input_size, hidden_size)

# Definir função de perda e otimizador
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Avaliação do modelo
with torch.no_grad():
    model.eval()
    y_pred = model(X_test_tensor)
    test_loss = criterion(y_pred, y_test_tensor)
    print(f'\nPerda no conjunto de testes sem treinamento: {test_loss.item():.4f

# Treinamento do modelo
num_epochs = 10000
for epoch in range(num_epochs):
    # Forward pass
    outputs = model(X_train_tensor)
    loss = criterion(outputs, y_train_tensor)

    # Backward pass e otimização
    optimizer.zero_grad()
    loss.backward()
```

```
    optimizer.step()

    if (epoch+1) % 1000 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# Avaliação do modelo
with torch.no_grad():
    model.eval()
    y_pred = model(X_test_tensor)
    test_loss = criterion(y_pred, y_test_tensor)
    print(f'\nPerda no conjunto de testes após: {test_loss.item():.4f}')
```

```
Perda no conjunto de testes sem treinamento: 2046188.6250
Epoch [1000/10000], Loss: 25.5772
Epoch [2000/10000], Loss: 25.3026
Epoch [3000/10000], Loss: 23.6792
Epoch [4000/10000], Loss: 21.9982
Epoch [5000/10000], Loss: 18.2380
Epoch [6000/10000], Loss: 15.5043
Epoch [7000/10000], Loss: 4.1860
Epoch [8000/10000], Loss: 6.4528
Epoch [9000/10000], Loss: 4.1939
Epoch [10000/10000], Loss: 3.7103

Perda no conjunto de testes após: 5.8763
```